Cranfield UNIVERSITY

Applications in Practical High-End Computing - Group Project

Assignment - "Workflow"

Test planning

Supervisor: Dr Stuart Barnes

Authors: Mateusz Gołąb
Csaba Kerti
Jakub Kiełbasa
Zsolt Kollarits

Course: CSTE / SETC

Table of contents

1 Summary

All software probably contains errors. To reduce errors and faults, we tried to use software driven development, and execute test operation after every steps of the software development project.

2 Requirement test

It is very important to find faults and misunderstandings at the early stage of a software development project as soon as possible. Furthermore, many projects fail because of the misinterpreted requirements. So it is utmost important to clarify the requirements with the customer.

The [1] web page gives many good ideas how to check and review the requirements of a software project. There is no standard method to execute this test. It is usually a review, made by the stakeholders and the customer should accept it if there is a change.

We checked the consistency of the user requirements, and are they clear enough. It should be clear for a designer or a programmer, who read it at the first time, and doesn't know the aims of the project. We had many assumptions at the beginning of the project, and we tried to discover them day by day. We contacted with the customer and clarified these assumptions. We read and reviewed the given document many times to clarify missing and hidden requirements.

2.1 Non-functional requirements

The non-functional requirements are very important too. Every non-functional requirement should contain concrete values to measure when the test will be executed. We reviewed the first version of the requirements, and we found some generally formulated sentences, which were modified later. The modifications were coordinated by the customer, to quantify the characteristics (e.g.: number of parallel users, supported operating systems).

We planned to execute the following non-functional test operations:

- Compatibility testing (network, operating system)
- Performance testing
- Security testing

3 Preparing Acceptance Test

3.1 Trace-ability Matrix for use-cases

The acceptance test should validate the user needs and requirements. It should be executed at the end of the project, but it is recommended to start the preparations at the beginning.

After we had produced the requirements we have created the use-cases using the UML. The Trace-ability Matrix is a good practice to check that is all the functional requirements are covered by use-cases.

We have created a Trace-ability Matrix to ensure that all requirements should have covered by uses-cases. The columns contain the requirements, while the rows contain the uses-cases. When a uses-case connects with a requirement, we sign it with a dot.

The Trace-ability Matrix is very usual tool to follow the requirements changes. By using them, it is easy to see which use-case should be changed.

	FU1	FU2	FU3	FU3.1	FU3.1	FU4	FU5
UC_U1 Log in						●	
UC_U2 Log out							
UC_U3 Stop Simulation							●
UC_S1 Run Simulation		●					
UC_S2 Upload Simulation parameters							
UC_S3 Download results of a Simulation							
UC_S4 Check status/log of Simulation				●			
UC_A1: Add new module to Workflow	●						
UC_A2: Remove module from Workflow	●						
UC_A3: Provide input/output metadata (XSD) of a Module							
UC_A4: Provide XML file with parameters and commands							
UC_A5: Provides XML file with recovery configuration							
UC_A6: Create new User							
UC_A7: Inactivate User							
UC_M1: Configure recovery mechanism					●		
UC_M2: Create backup			●				
UC_M3: Recover			●				

1. Table – Trace-ability matrix for use-cases

Where the user requirements are the followings:

- FU1 User can add/remove arbitrary number of modules into workflow.
- FU2 User can run simulation with uploaded parameters.
- FU3 Recovery system: possibility to restart workflow from the last stable/good point when system crushes.
    - FU3.1 Monitoring of errors: Users can see the exact location of failures.
    - FU3.2 Flexible recovery policy (depending of the expected time of execution, we decide to store data before/after a module or after each iteration)
- FU4 Many users have the possibility to connect to the system simultaneously. But there is only one running program at the same time (users requests' go to queue - serial workflow).
- FU5 User can stop simulation.

3.2 Test cases

We created test cases to test the correctness of each operation and to examine how they affect each other. We made many test case variants for each operation to reach better test coverage.

We chose the following form to describe each test case (we will fill it with concrete values)

Title	Content
Test Case ID	
Description	
Input	
Steps	1. 2. 3…
Dependencies	
Expected result	

Following test cases are connected to general User activities:

Test Case ID	Description
TC_0001	login with a valid scientist login name and password
TC_0002	login with a valid administrator login name and password
TC_0003	try to login with an invalid login name
TC_0004	try to login with valid login name with invalid password
TC_0005	logout working properly
TC_0100	scientist tries to stop own simulation
TC_0101	scientist tries to stop not own simulation
TC_0102	administrator tries to stop arbitrary simulation
TC_0103	user tries to stop an empty workflow

Test cases connected with scientist activities:

Test Case ID	Description
TC_0200	scientist tries to start a simulation with appropriate parameters
TC_0201	scientist tries to start a simulation with invalid parameters
TC_0202	scientist tries to start a simulation with empty parameters
TC_0203	scientist tries to start a simulation when the WorkflowQueue empty
TC_0204	scientist starts a simulation when the WorkflowQueue doesn’t empty
TC_0300	scientist tries to download the result of the simulation
TC_0301	scientist tries to download the result when there is no result
TC_0400	scientist tries to check the status/log of own simulation
TC_0401	scientist tries to check the status/log of not owned simulation
TC_0402	administrator tries to check the status/log of any simulation

Test cases connected with administrator activities:

Test Case ID	Description
TC_0500	add new module to an empty workflow
TC_0501	add new module before an existing module
TC_0502	add new module after an existing module
TC_0503	insert module to an existing workflow
TC_0504	add new start module, when the WorflowQueue is not empty
TC_0505	add new module when a simulation is not running
TC_0506	add new module when a simulation is running
TC_0600	remove an alone module from the workflow
TC_0601	remove module from the middle of the workflow
TC_0602	remove module from the end of the workflow
TC_0603	remove module from the beginning of the workflow
TC_0604	remove module when the simulation is running
TC_0605	remove module from the beginning when the WorflowQueue is not empty
TC_0700	provide valid input and valid output metadata of a Module
TC_0701	provide invalid input and invalid output metadata of a Module
TC_0702	provide valid input and invalid output metadata of a Module
TC_0703	provide invalid input and valid output metadata of a Module
TC_0800	provide XML file with parameters and commands
TC_0801	provide inappropriate structure of XML with parameters and commands
TC_0900	provide XML file with recover configuration
TC_0901	provide inappropriate structure of XML with recover configuration
TC_1000	try to create an user which is already existing in the system
TC_1001	try to create an user whit invalid details
TC_1002	try to create a scientist
TC_1003	try to create an administrator
TC_1100	inactivate an administrator
TC_1101	inactivate a scientist
TC_1102	inactivate a scientist who have a simulation in the WorkflowQueue
TC_1103	inactivate a scientist who have a running simulation
TC_1104	inactivate a user who is online in the system

Test cases connected with WorflowManager, RecoveryManager and DatabaseManager

Test Case ID	Description
TC_1200	Workflow/Recovery Manager start to work without configuration (conf.XML)
TC_1201	Workflow/Recovery Manager start to work with configuration
TC_1300	Module execution , when the Recovery Manager should create a backup
TC_1301	Module execution , when the Recovery Manager should not create a backup
TC_1302	Test that if there was not backup under the iteration, then at the end of the iteration the RecoveryManager should create it
TC_1303	Try to create backup when the Database is unreachable
TC_1400	The system recover the last stable point successfully
TC_1401	The system can't recover the last stable point
TC_1402	Artificial execution where after the first failed execution and the successfully recovered stable point the execution is successful.
TC_1403	Artificial execution where after the first two failed execution and the successfully recovered stable point the execution is successfully
TC_1404	Artificial execution for what happened when all the recovery attempt fail.

Example Details of the TC_1402 test case:

Title	Content
Test Case ID	TC_1402
Description	Artificial execution where after the first failed execution, and the successfully recovered stable point, the execution will successful.
Input	Appropriate module which fails at the first time and successful at the second time.
Steps	1. Administrator creates a new workflow with this module as a start module. 2. Administrator sets the appropriate XML and XSD files. 3. Scientist starts the simulation with appropriate parameters.
Dependencies	
Expected result	The execution of the module should fail. After the Recovery Manager recovery the workflow and the second execution should be successful.

3.3 Trace-ability Matrix for test-cases

The Trace-ability Matrix is also usable to check that at least one test-case exists for every use-cases. The columns contain the use-cases, while the rows contain the test-cases. When a test-case connects with a use-case, we sign it with a dot.

	UC_U1	UC_U2	UC_U3	UC_S1	UC_S2	UC_S3	UC_S4	UC_A1	UC_A2	UC_A3	UC_A4	UC_A5	UC_A6	UC_A7	UC_M1	UC_M2	UC_M3
TC_0001	•																
TC_0002	•																
TC_0003	•																
TC_0004	•																
TC_0005		•															
TC_0100			•														
TC_0101			•														
TC_0102			•														
TC_0200				•	•												
TC_0201				•	•												
TC_0202				•	•												
TC_0203					•												
TC_0204					•												
TC_0300						•											
TC_0301						•											
TC_0400							•										
TC_0401							•										
TC_0402							•										
TC_0500								•									
TC_0501								•									
TC_0502								•									
TC_0503								•									
TC_0504								•									
TC_0505								•									
TC_0506								•									
TC_0600									•								
TC_0601									•								
TC_0602									•								
TC_0603									•								
TC_0604									•								
TC_0605									•								
TC_0700										•							
TC_0701										•							
TC_0702										•							
TC_0703										•							
TC_0800											•						
TC_0801											•						
TC_0900												•					
TC_0901													•				
TC_1000													•				
TC_1001													•				
TC_1002													•				
TC_1003													•				
TC_1100														•			
TC_1101														•			
TC_1102														•			
TC_1103														•			
TC_1104														•			
TC_1200															•		
TC_1201															•		
TC_1300																•	
TC_1301																•	
TC_1302																•	
TC_1303																•	
TC_1400																	•
TC_1401																	•
TC_1402																	•
TC_1403																	•
TC_1404																	•

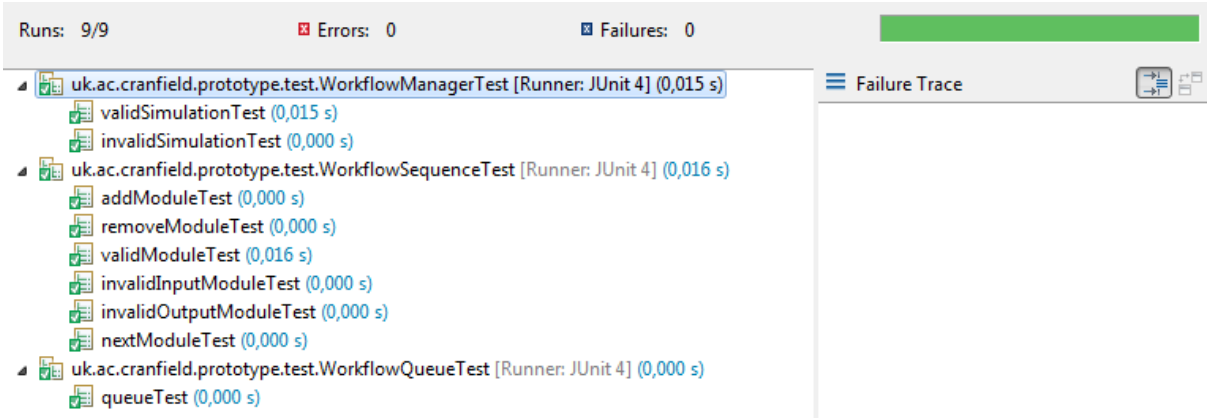
2. Table - Trace-ability matrix for test-cases

4 Unit test

We choose the Java programing language for the implementing our system. JUnit is an easy to use environment for Java, to create repeatable tests. So it was convenient to choose the JUnit as a testing environment.

We planned to create JUnit testes for every unit and component.

Many mock classes have been created to ensure the excitability of the prototype. Furthermore these mock classes are very important to execute the unit testes. We have created unit testes for the implemented classes which provide full functionality. Few testes have been created in each unit test to cover the every class functionality. The following diagram shows the result of unit test execution.



5 Integration Test

We plan to test the bigger components of the application, and examine how they work together. The modules which we plan to test in the integration test:

- WorkflowManager
- RecoveryManager
- WorkflowSequence

JUnit is not limited for unit tests, so we decided to use JUnit where it is a possible in the system.

A few scenarios what we plan to examine:

1. The Workflow gets knowledge about the unsuccessful execution. It asks the RecoveryManager to get the last stable point, who forwards it to Database Manager. Finally the WorkflowManager should get the last backup from the DatabaseManager through the RecoveryManager.
2. After a successfully execution, if the RecoveryManager decide to store the result it sends it to the DatabaseManager and the backup will be store in the database.

6 References

[1] http://www.softwaretestinghelp.com/how-to-test-software-requirements-specification-srs/, 2012

[2] Dorothy Graham, Erik van Veenendaal, Isabel Evans, Rex Black : Foundations of software testing

[3] Brian Hambling : "SOFTWARE TESTING - An ISTQB–ISEB Foundation Guide", 2nd Edition, 2010

[4] Rex Black, Jamie Mitchell: Advanced Software Testing Vol. 3, 2011